72

Popular Computing

The world's only magazine devoted to the art of computing.

March 1979 Volume 7 Number 3

1	8	9	2	7	10
16	17	24	15	18	23
25	32	33	26	31	34
4	5	12	3	6	11
13	20	21	14	19	22
28	29	36	27	30	35

Solution Evolution – 1

homework 7

In a normal 365-day year, January 4 at 15:36 (that is, 3:36 P.M.) marks the 1% point--the exact time when 1/100 of the year has passed. Each such successive percentile point in the year is $3^{d}\ 15^{h}\ 36^{m}$ later. With the days in the year numbered from 1 to 365, the accompanying flowchart suggests a scheme for generating all 100 of the percentile points in the year.

(A) Modify the given scheme (or devise a new one) to output the month, day, and hour properly. Thus, the time denoted by

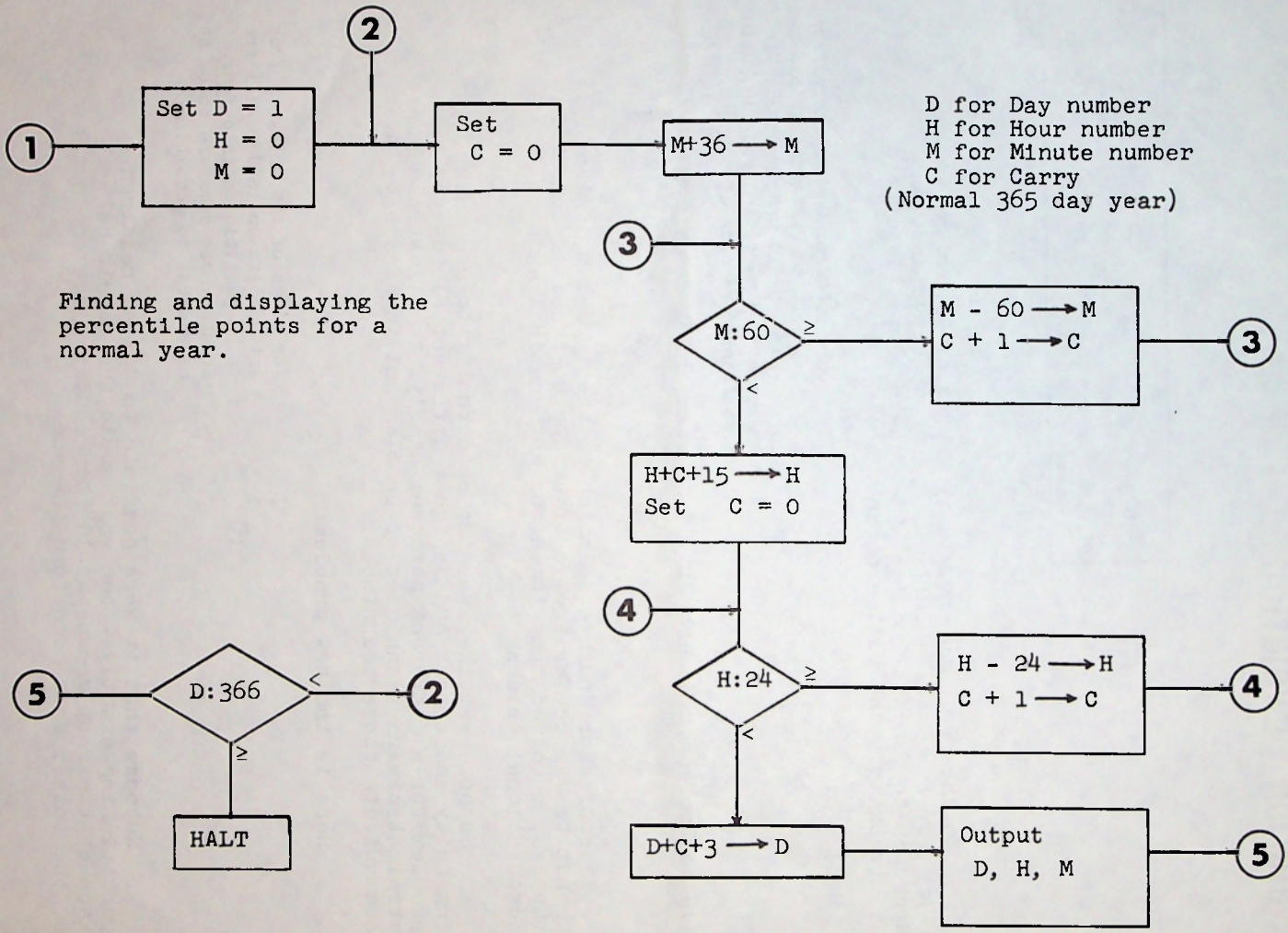
day 44 hour 19 minute 12

should appear as 2-13-7-12-2, to represent February 13 at 7:12 P.M.

(B) Devise a corresponding scheme to find the 100 percentile points in a 366-day year.

Publisher: **Audrey Gruenberger**
Editor: **Fred Gruenberger**
Associate Editors: **David Babcock**
Irwin Greenwald
Patrick Hall
Contributing Editors: **Richard Andree**
William C. McGee
Thomas R. Parkin
Edward Ryan
Art Director: **John G. Scott**
Business Manager: **Ben Moore**

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 per year. For all other countries, add $3.50 per year. Back issues $2.50 each. Copyright 1979 by POPULAR COMPUTING.



D for Day number
H for Hour number
M for Minute number
C for Carry
(Normal 365 day year)

Finding and displaying the percentile points for a normal year.

Solution Evolution – 1

> *After a program is written, debugged, and thoroughly tested--only then do we know just how it should have been written.*

In the two-part article by Professor Donald E. Knuth ("Are Toy Problems Useful?" in issue number 46 and "An Approach to Floyd's Problem" in issue number 47) he used as his example the following problem:

P — The square roots of the integers from 1 to 50 are to be partitioned into two parts whose sum is nearly equal; find the best such partition you can, using less than 10 seconds of computer time.

due to Professor Robert Floyd* of Stanford University.

Professor Knuth's solution to that problem, which he admits may be far from optimum due to the time constraint imposed, involved many ingenious mathematical and computational tricks.

Suppose we take away the opportunity to use such ingenuity and tricks and design a different Floyd-type problem for a different purpose; namely, to illustrate some straightforward techniques of problem solving and ways to communicate those techniques.

Here is the new problem:

Q — The sines of the integers from 1 to 36 (taken in degrees) are to be partitioned into four groups whose sums are to be as near-equal as possible.

Suppose that it were possible to do the required partitioning exactly--then what would the result be? Each of the four groups would then contain just one quarter of the total, so we can form the sum:

* Professor Floyd was named the recipient of the Turing Award (ACM) for 1978.

$$
\begin{aligned}
\sin(1) &= .0174524064372 \\
+\sin(2) &= .0348994967022 \\
+\sin(3) &= .0523359562427 \\
&\vdots \\
+\sin(36) &= .587785252292 \\
\text{total} &= 11.2361350322 \\
1/4\ \text{total} &= 2.80903375805
\end{aligned}
$$

But since most of those 36 numbers to be added are transcendental, it is not likely that a perfect solution is possible. We do not even know, a priori, that a good solution (whatever that may mean) is obtainable, short of trying every possible combination (which is some enormous number, of the order of 2 million). There is no reason to believe that a good solution will have 9 of the numbers in each group.

Well, pending the arrival of a flash of insight, a la Knuth on the earlier problem from Floyd, we should do something.

As a crude first cut at a solution, suppose we simply distribute the 36 numbers in rotation among the four groups, just to see what happens. We get this:

group A:	2.58880024
group B:	2.736185671
group C:	2.882737634
group D:	3.028411487

Intuitively, we must feel that this is a poor solution. By assigning each of the 36 numbers in sequence among the four groups, group D naturally gets the largest of each set of four, taking the sines in their natural order, which is monotonically increasing.

Let's try a minor modification (pictured on the cover of this issue). We assign the 36 numbers arbitrarily in a slightly different order of rotation, with this result:

group A:	2.78498223703
group B:	2.80158631090
group C:	2.81733699444
group D:	2.83222948984

This appears to be an improvement, but nothing to brag about. We could try two other approaches:

(1) Assign sin(1) to A, sin(2) to B, sin(3) to C, and sin(4) to D to start. Then add sin(N) to that variable that has, at the moment, the smallest sum (N running from 5 to 36).

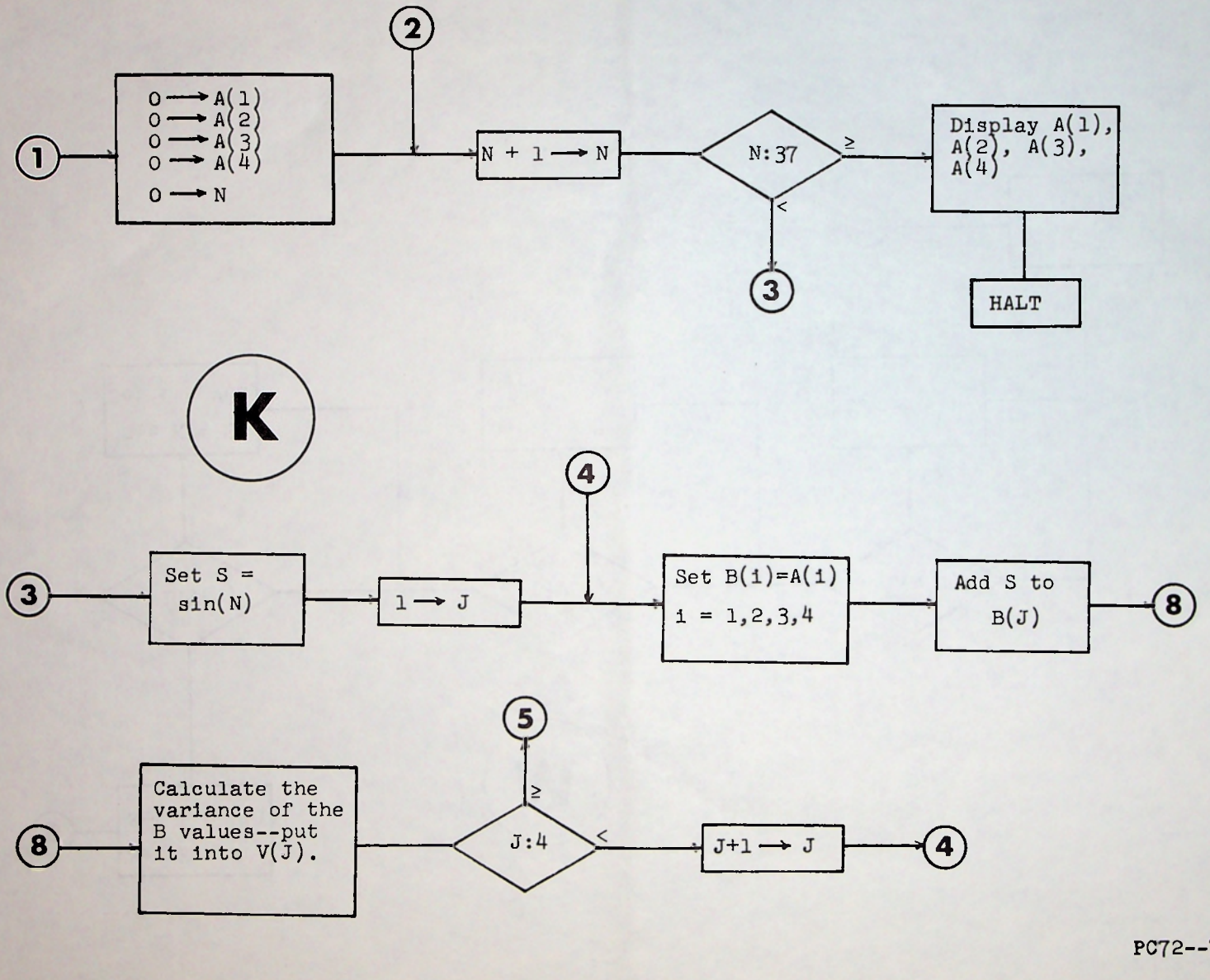
(2) Start as in (1) above. Add sin(N) to that variable that will make the variability among A, B, C, and D the least.

The first of these approaches is rather simple minded; each new sine is added to the variable that seems to need it the most at that time. The second approach seems to be much more sophisticated, and arises naturally from the observation that we have not considered any criterion for what might constitute a good solution to the problem, or how to rank one solution as better than another.

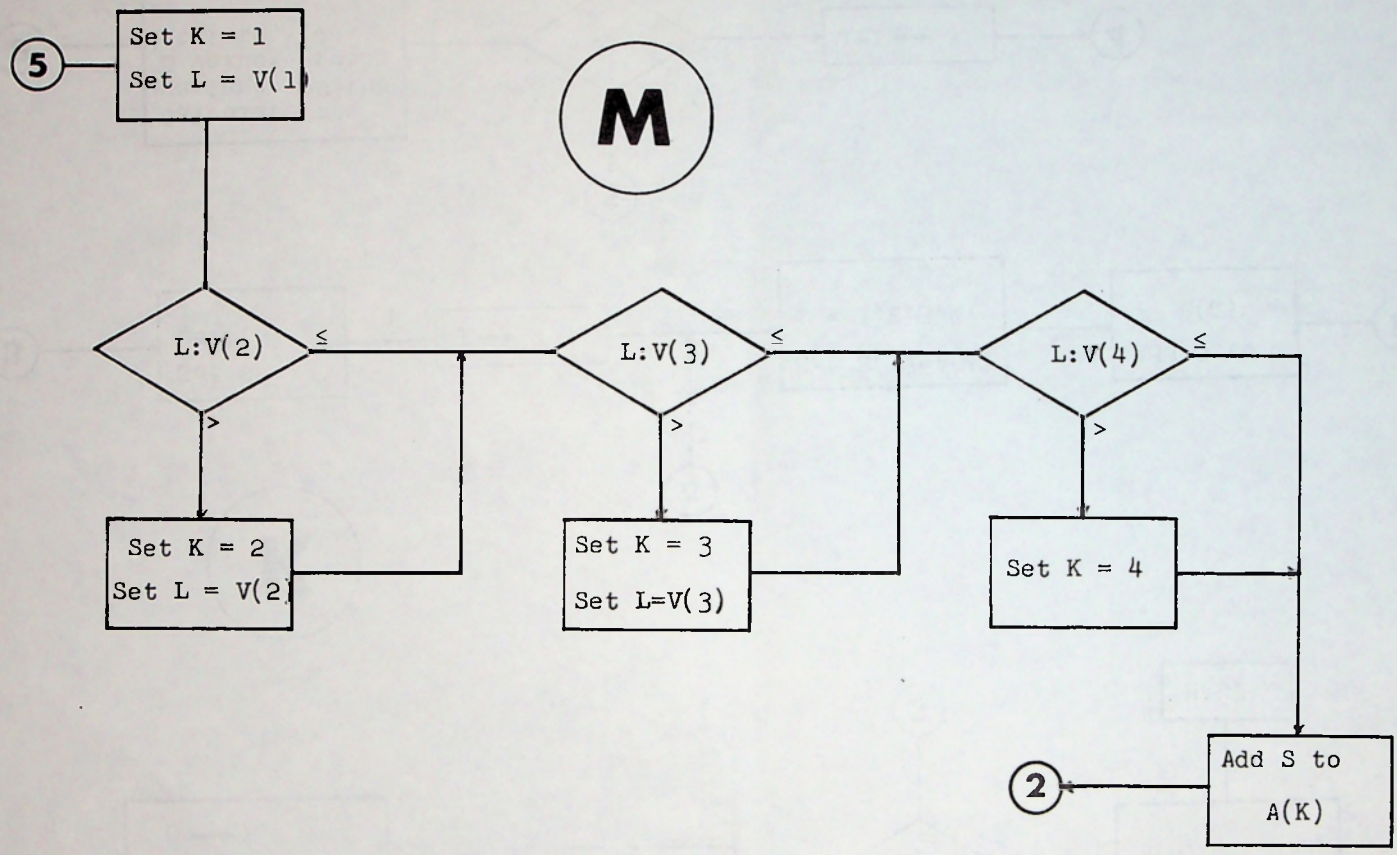
Let's consider the second of those approaches. The scheme to implement it is shown in Flowcharts K and M. The notation has been altered slightly: the four variables are called A(1), A(2), A(3), and A(4) here. At any given stage, we will copy the A array in an array B (Reference 4), and add the next element into B(1) and calculate the variance of the B array, calling it V(1). This process is repeated four times. The second time, the A array is again copied into the B array; the next element is added to B(2); the variance is calculated and stored as V(2); and so on, to form V(3) and V(4). The logic of Flowchart M then selects which of the four variances is the smallest, and the new element is then added into the appropriate variable in the A array.

Flowcharts K and M attempt to communicate the above approach to the problem. But now we have come to the point of this article; namely, how good are flowcharts at such communication? There is a large and vocal school that says that flowcharts are a dead tool, and were never any good anyway. Another group says that there must be a better tool, but they cannot agree on what it might be. Still another group plays endlessly with various ways in which flowcharts might be drawn.

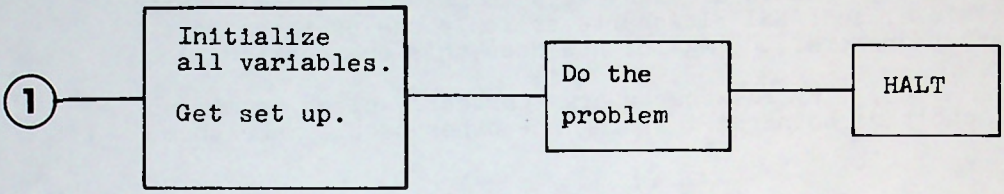
Some of the notation of flowcharts has been standardized--the least important part; namely, the mechanical structure of the flowchart, and trivial things like the exact shape of the boxes. At that, the published standard includes some 40 differently shaped boxes when, for most purposes, half a dozen shapes would be more than adequate.


1
2
0 → A(1)
0 → A(2)
0 → A(3)
0 → A(4)
0 → N
N + 1 → N
N:37
≥
<
3
Display A(1), A(2), A(3), A(4)
HALT
K
3
Set S = sin(N)
1 → J
4
Set B(i)=A(i) i = 1,2,3,4
Add S to B(J)
8
8
Calculate the variance of the B values--put it into V(J).
5
≥
J:4
<
J+1 → J
4

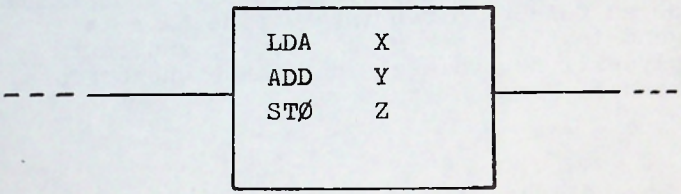
PC72--7


5
Set K = 1
Set L = V(1)
M
L:V(2)
≤
>
L:V(3)
≤
>
L:V(4)
≤
>
Set K = 2
Set L = V(2)
Set K = 3
Set L=V(3)
Set K = 4
2
Add S to
A(K)

There is no standard whatever for what goes <u>into</u> the little boxes, nor even any agreement as to what level a flowchart should aim for. Thus, in our textbooks, we have every level between the highest possible:



and the lowest possible:



There is not even a common notation; the use of the right-pointing arrow to mean "replaces" (or the left-pointing arrow to mean "is replaced by") or the use of the colon to stand for "compare" is not widely adopted. Some users of flowcharts insist on having every decision made binary, and preferably with the exits labelled "true" and "false."

Other objections to the ANSI standard of flowcharting (or to flowcharting in general) are these:

(1) "I learned a different style of flowcharting which I like better, so I'll sulk at using yours."

(2) "Flowcharts are a bore; it's more fun to plunge into coding without that nuisance."

(2B) "I can always go back and make a flowchart from my running code."

(2C) "In fact, there are computer programs for sale to do just that for me."

(3) "When I correct or modify the program, then the program is no longer in agreement with the flowchart..." (How should this one be finished?) "...and I'm too lazy ever to go back to alter the flowchart."?

(4) "There are many ways to make a problem solution visible, and ANSI flowcharts are only one of those ways and therefore..." (Again, how does this one continue?)

(5) "My flowcharts are perfectly clear to me, so I can't be bothered learning how other people make them."

One wonders how any other organized group--say, the construction trades--would have survived with such rampant permissiveness as this. Blueprints are to have their title and legend in the lower right corner, and that industry simply will not tolerate the yahoo who prefers to put it somewhere else.

But we have digressed. Go back to Problem Q. It would be nice to produce some kind of reasonable solution to it (how about a Monte Carlo attack, in which the 36 numbers are distributed at random among the four variables, and this procedure is continued until the variability becomes suitably low?)

But solving the problem is peripheral to the point here: How do you intend to communicate your solution to others (and to yourself, in case there are changes or corrections to be made)?

What are the possible choices? Ultimately, the only precise expression of a solution is the program itself, but despite the claims made for the readability of programs (claims that have been made at one time for nearly every computer language written), it is difficult to picture a solution that is expressed in such terms. How about narrative flowcharts? Or pseudocode?

We ought to have some tool for organizing the solution to a computer problem; for communicating the essence of that solution to others; and for documenting the solution. If not flowcharts--what?

BOOK REVIEW

Probability, Statistics, and Queueing Theory

by Dr. Arnold O. Allen

The student of queueing theory is often handicapped by inadequate preparation in probability theory, upon which queueing theory rests. Either he has never had a course in probability, or he has but has forgotten everything he ever knew. Authors of queueing theory books (such as L. Kleinrock's Queueing Systems) sometimes attempt to solve this problem by including a condensed "crash course" in probability in an appendix, hoping that the student will read the appendix before tackling the body of the text. Students rarely do this, and consequently are unable to fully grasp the mathematical basis of queueing theory, or to apply it in contexts different from those covered by the author.

Arnold Allen of IBM's Systems Science Institute has taken a different approach to this problem in his text Probability, Statistics, and Queueing Theory. This text is first and foremost a text on probability and statistics, with six out of eight chapters and almost two-thirds of the total page count devoted to these subjects. Queueing theory and its applications in computer science occupy a lesser role in the book, and are introduced only when the reader has been thoroughly exposed to probability fundamentals. The book would therefore serve well in a computer science or engineering curriculum where a separate course in probability or statistics cannot reasonably be a prerequisite.

The book is divided into three parts: (1) Probability, (2) Queueing Theory, and (3) Statistical Inference. Part 1 introduces the student to the notion of probability, and takes him through such topics as sample spaces, events, combinatorial analysis, random variables, discrete and continuous probability distributions, and stochastic processes. Included in the discussion of distributions are the Erlang distributions and the hyperexponential distribution, which figure prominently in queueing systems.

The discussion of stochastic processes in Part 1 leads naturally to Part 2, where probability theory is applied to the analysis of queueing systems. Queueing systems are classified as birth-death systems, embedded Markov chain systems, and priority queueing systems. These systems are in turn applied to various computer modeling problems, such as round-robin time sharing, multiprogramming, and interactive terminal computer access. Networks of queues, cmmon in computer modeling, are given extensive treatment.

Part 3 covers traditional statistics, including discussion of parameter estimation, confidence intervals, and hypothesis testing. In most real-life modeling problems, the parameters of the probability distributions underlying the phenomenon being studied are not known, and a knowledge of statistics is indispensable in such situations.

The book uses a compact, mathematical style (a knowledge of calculus is recommended). The exposition and notation are unusually clear. The mathematical rigor is relieved by frequent use of jocular comments and examples. Problems are couched in terms of hypothetical enterprises with names like Big Bored Securities and Kamakazy Airlines, and each chapter concludes with a few "student sayings" which the author has collected over the years of teaching this material (example: "Our observation of Nancy's distribution has given us many fine moments.").

The author follows D. Knuth's practice of grading the exercises by complexity. An example of an elementary exercise is: Fortran and Algol coding sheets are mixed up in a drawer; what is the smallest number that need be selected sight unseen to guarantee getting two of the same kind? (This is an adaptation of the old brown-and-blue-socks-in-a-drawer puzzle.) An advanced exercise is: Customers arrive randomly (during the evening hours) at the Kittenhouse, the local house of questionable services, at the average rate of five per hour. Service time is exponential with a mean of 20 minutes per customer. There are two servers on duty. If the house is raided, how many customers will be caught, on the average? (Answer: 5.45.)

A nice feature is an appendix containing APL programs for doing the common calculations in queueing studies, such as computing the statistics for the machine repair queueing model. The author has also thoughtfully included an appendix of one-page summaries of the formulas for the various queueing systems (M/M/1, M/M/n, etc.).

This is a well-executed, useful book which should be in every computer scientist's library.

--Reviewed by W. C. McGee

Solution Evolution—2

> *After a program is written, debugged, and thoroughly tested--only then do we know just how it should have been written.*

The purpose of this article is to describe how an efficient problem solution developed, starting with a straightforward--but inefficient, inelegant, and crude--solution.

The problem involved here is the following:

Three variables, A, B, and C are to be progressively incremented with the mantissas of successive square roots. The initial values of the three variables are thus:

A = .23606797749 (from $\sqrt{5}$)

B = .41421356237 (from $\sqrt{2}$)

C = .73205080756 (from $\sqrt{3}$)

Each new mantissa (the next one is .44948974278) is to be added to the variable whose contents at that time is the smallest (thus, the mantissa of the square root of 6 is to be added to A).

What we want at each stage is the variance of the contents of A, B, and C, and specifically we want to record successively lower lower values of the variances. The variance is calculated as:

$$\frac{N\Sigma X^2 - (\Sigma X)^2}{N(N-1)}$$

which in this case becomes:

$$\frac{3\Sigma X^2 - (\Sigma X)^2}{6}$$

[You now have the problem--would you care to try your hand at developing a neat, elegant solution before reading on?]

The general plan seems quite clear. Generate the next value of N, take its square root, and get rid of the integer part; this furnishes the required mantissa. Find which of A, B, or C has the lowest total at the moment, and add the mantissa to it. Calculate the variance, using the given formula, and output it. Flag the output in some way at each appearance of a new lower variance.

In the interest of getting some results (and to get a feel for the problem), some shortcuts to the above plan were adopted:

1. On most machines (that is, in most programming languages), getting the fractional part of a number is an awkward procedure. The first solution here would be made interactive, to display the square root of N, so that the integer part could be subtracted manually. Thus we can avoid straining the brain by substituting a time-consuming manual step.

2. If we seek to find how the variance behaves (as opposed to recording the actual variance), then we can neglect the denominator of 6 and use only the numerator of the variance formula.

So we have a quick and dirty solution, outlined in Flowchart F.

The variance calculation is this:

$$3(A^2 + B^2 + C^2) - (A + B + C)^2$$

which, with a little algebra, can be transformed into:

$$(A - B)^2 + (A - C)^2 + (B - C)^2$$

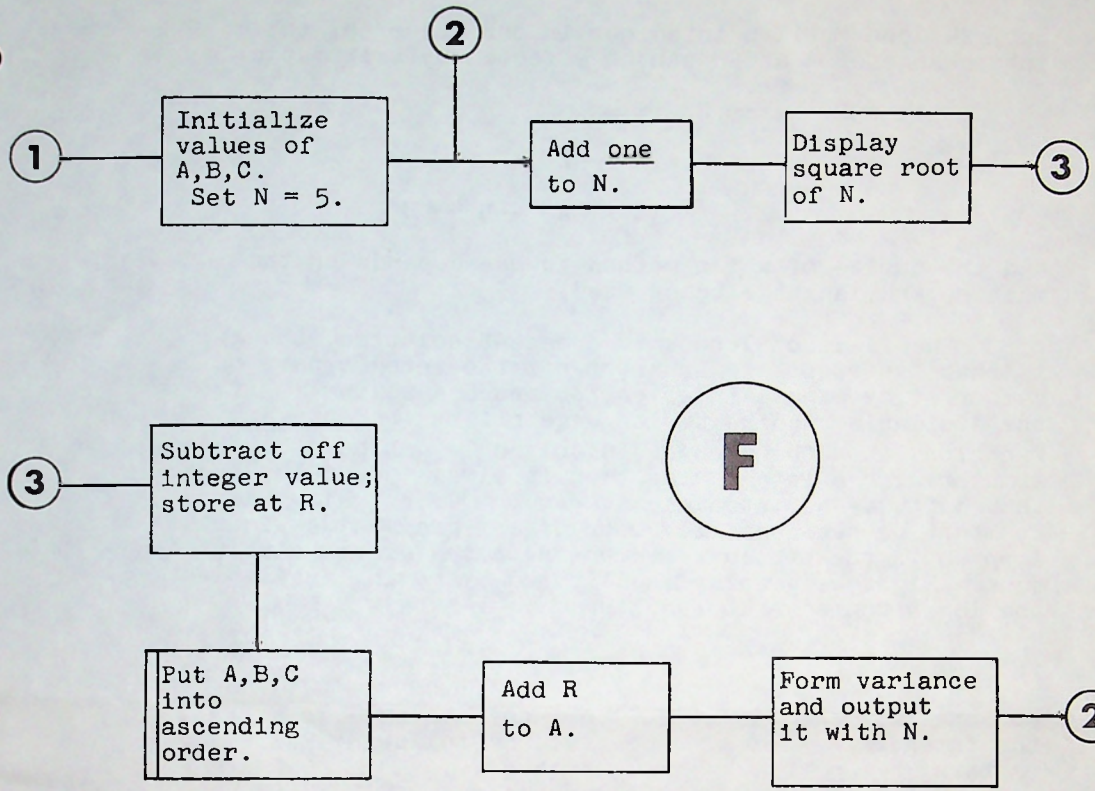
which is easier to calculate, and saves both instructions and storage.

The operation indicated at Reference 3 in the flowchart is simply the first thing that came to mind; if the value R is to be added to the smallest of A, B, and C, it seems natural to sort them into ascending order and then add into A. Here again there are choices to be made. Any scheme for sorting three things will involve three comparisons and interchanges. How shall the interchanges be made? The obvious way, found in the textbooks, is to use a spare storage word, thus:

Move A to SPARE

Move B to A

Move SPARE to B



Text continues on page 16

...Continued from the review on the last page...

There is a photograph of a module from the IBM 704 (with the photo credited to the Digital Equipment Corp.), attributed to 1952. There are many such references, each just a bit off. In 1952, IBM was wondering if they could unload some 701's--the 704 was still a dream.

There are good things about Arbib's book. On the "society" side, he is ahead of his competitors in the clarity of his exposition and in his choice of exercises (the title page credits Jonathan V. Post for supplying many exercises). There is a glossary at the end of each chapter with definitions apparently written by the author (i.e., not lifted from some "official" glossary) which are generally excellent, although not very technical. Arbib's writing is lucid, literate, and well organized.

We need books on computers and society. We still need one that has both parts well done.

But, as John Motil pointed out in our issue 56, the interchange of A and B can be effected by arithmetic:

$$B - A \longrightarrow B$$

$$B + A \longrightarrow A$$

$$A - B \longrightarrow B$$

and the choice of which method to use depends on the machine and language being used.

The real bottleneck to a decent solution, though, is that "shortcut" to subtract off the integer part of each root by manual intervention and eyeballing. If the available machine or language offers the integer function, then the problem is solved. On the other hand, simulating the integer function is always possible, even though it may be extremely awkward. In some languages, it might be necessary to subtract integral values in a loop until the procedure zeros in on the exact value (this scheme might be greatly facilitated by having available the logarithm base 10 function).

(On one programmable calculator, one sequence to extract the integer part of a number was the following weird pattern:

Left parenthesis
Store
Minus .5
Right parenthesis
Fix zero
Exponent
Inverse exponent
Inverse fix.)

But all this is heading down the wrong path (and going down blind alleys is characteristic of most problem solutions). Every one of the 55 values of N from 729 through 783, for instance, will have 27 subtracted from its square root. Thus, for a considerable distance, there would be no need to calculate the integer function--it's 27 in that range. This notion can be extended throughout the whole range of N values, based on the familiar fact that the first differences of the squares are the odd numbers. By predicting the integer amount of each square root, this part of the larger problem is made to go away.

To implement the idea, however, it is necessary to get started properly (another common element of problem solutions). The variables A, B, and C are initially set to the mantissas of the square roots of 2, 3, and 5, respectively. We must add in the squre roots of 6, 7, and 8 in order to arrive at a perfect square. With a little work, we can determine these starting values for the three variables:

A: 1.513984845029

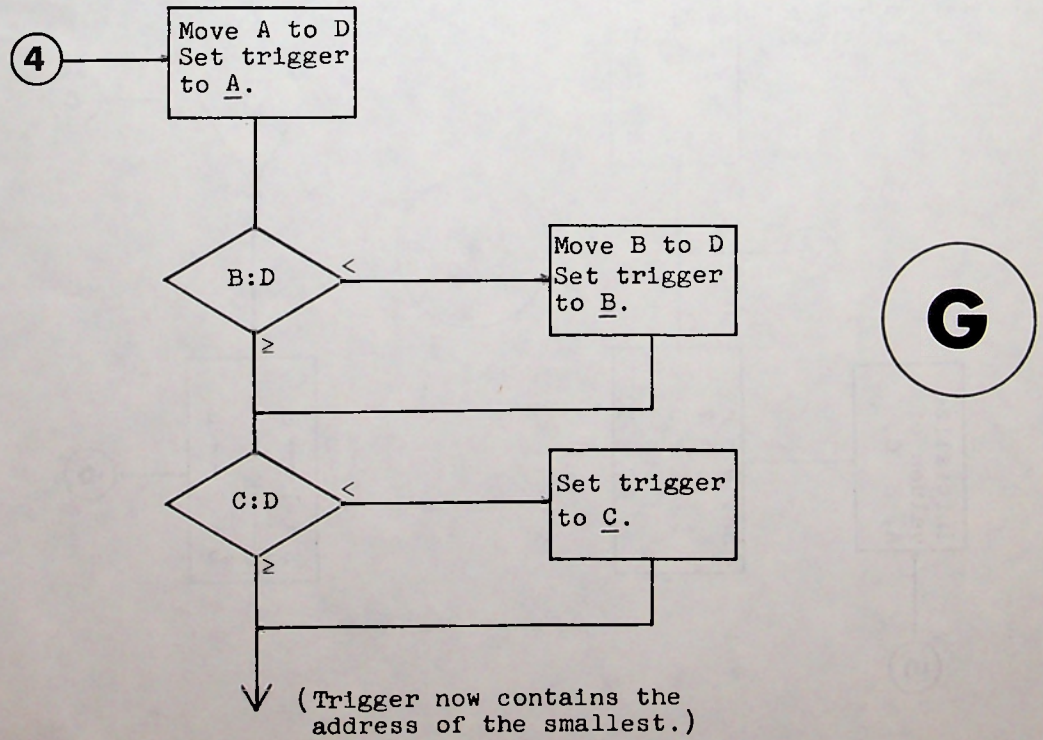
B: 1.059964873438

C: .732050807569

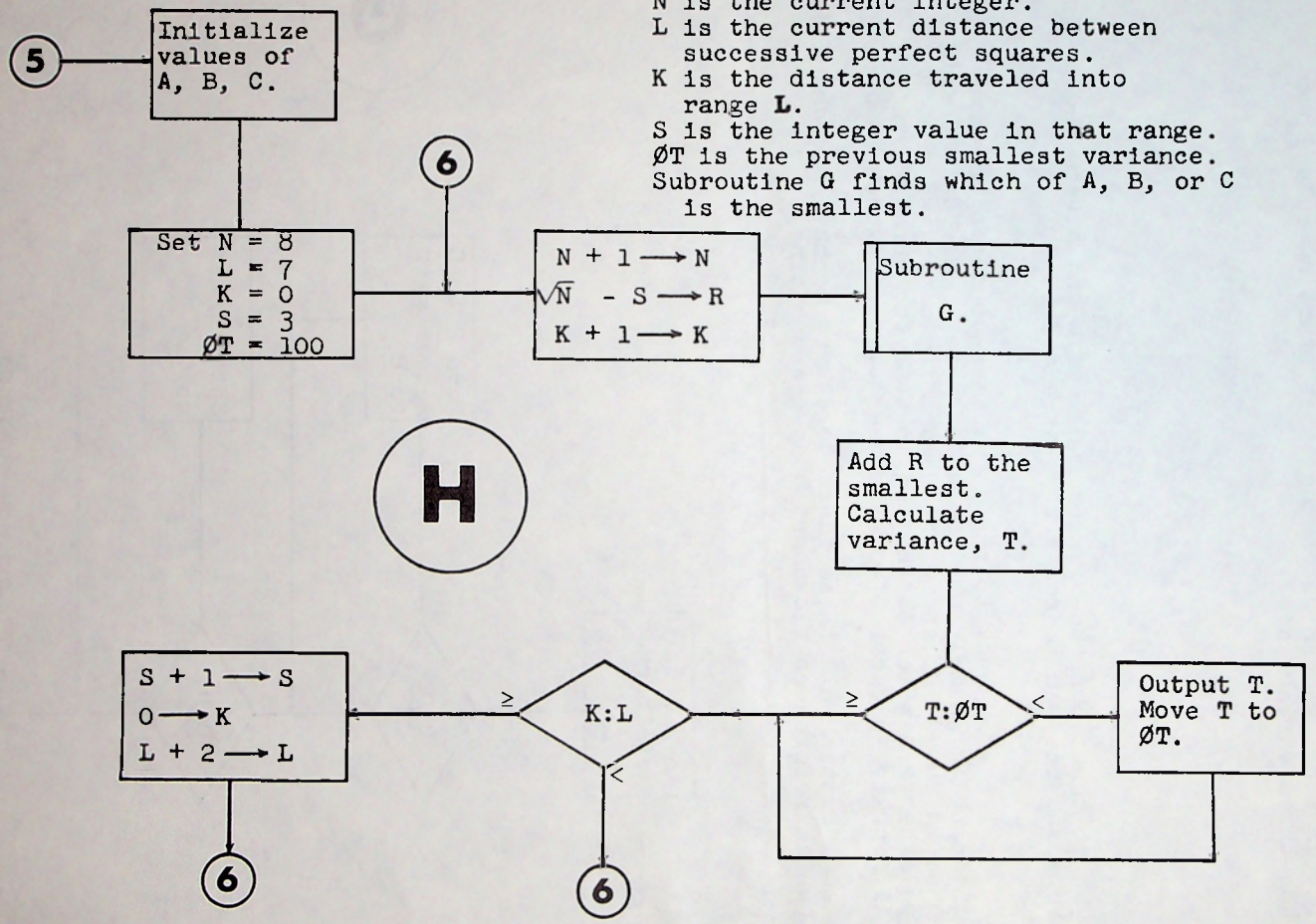
(as it turns out, the square roots of 6 and 8 were added to A and the square root of 7 was added to B.)

Let's go back to the other time-consuming part of the problem; namely, the sorting phase. We don't really need to put A, B, and C into ascending order; we want only to ascertain which is the smallest of the three. This difficulty is cleared up with the logic of flowchart G.

Putting all these ideas together, we have flowchart H (which references flowchart G as a subroutine).



N is the current integer.
L is the current distance between successive perfect squares.
K is the distance traveled into range L.
S is the integer value in that range.
ØT is the previous smallest variance.
Subroutine G finds which of A, B, or C is the smallest.



The problem used as an illustration here is totally artificial, but has some interesting features:

(a) The problem is serial in nature. It cannot be solved faster on two machines; each stage depends on all the preceding stages.

(b) Thus, a restart procedure must record carefully the values of all the variables. If I have carried the process to N = 6000 but do not furnish you the values of A, B, and C at that point, you cannot carry on where I left off.

(c) This problem may possibly be sensitive to precision levels. My results show the following values of the variance, as successively lower values appear:

N	[Variance]
6	.176809
18	.071521
29	.014508
54	.012894
86	.007127
634	.006228
686	.003234
738	.003213
1949	.001910

with the calculations made with 12-digit precision. Notice that the problem, by its very nature, demands ever-higher precision levels. For example, with a 12-digit system, when we reach N = 100 000, the mantissa of the square root (.227766016) is good to only 9 digits. At N = 10 000 000, the mantissa (.27766016) is good to only 8 digits, and so on. If the problem is extended far enough, the precision level used will dictate the results. Intuitively, the table shown above can be extended indefinitely, since the mantissas of square roots become arbitrarily small. For example:

$$\sqrt{1522757} - 1234 = .00040518$$

$$\sqrt{1522758} - 1234 = .00081037$$

and so on (taking successive integers just after a perfect square). Thus, it seems reasonable that the variance between A, B, and C can be reduced arbitrarily also. We solicit extensions to the table given above.

Book Review...

Computers and the Cybernetic Society

by Michael Arbib

Academic Press 1977, 494 pages, hard cover.

We have here another "Computers and Society" book. We can probably use quite a few such books, with differing views of the interaction between computers and society. To be viable, they should be written by someone who knows something about computers. Arbib is strong on the cybernetic society, but weak on computers. He starts the book by saying:

> *There seem to be two extremes in books on computers and society. At one extreme is the book that loads the reader down with so many details about how computers work that they have no time or energy left for the social implications. At the other extreme is the book that tells readers so many different effects that computers are having on their lives that they can see no pattern in the details. This book tries to strike a balance.*

He has a small working knowledge of computers and computing, and he gets all excited about robots, and CAI, and automation and democracy, and on and on. He sees computers at the heart of everything. Unfortunately, his view is clouded with errors. Whenever he shifts from sociology (at which he seems at home) to computers, he seems to be on unfamiliar ground, and he then flounders around trying to explain things he is not sure of. His treatment of a subject like sorting is painful to read.

The Name Index (which is one of the good features of the book) does not include von Neumann, Eckert, Mauchly, Babbage, Ada Augusta, or Wilkes, but does include three references to T. H. Nelson and four references to M. A. Arbib. Such a survey can be used to calibrate the book.

Please turn to page 15...